Applied Math Exascale Workshop Position Paper: A Challenge Problem Competition and Multi-Algorithm Optimizations

W.D. Henshaw, LLNL, henshaw@llnl.gov,
J.W. Banks, LLNL, banks20@llnl.gov,
LLNL-PROP-635760

April 29, 2013

1 Challenge problem competition

Proposition 1. *The DOE community should develop a set of* exascale challenge problems *that are representative of important DOE applications (e.g. accelerators, ground-water, climate, MHD, combustion, ...). Multiple teams should be funded to solve these challenge problems. The teams will compare results and the team with the "best" results will be awarded a significant "prize".*

Here is an example of a challenge problem from accelerators:

- Solve the time-domain Maxwell's equations in an complex accelerator cavity for the the wake-field produced by a charged pulse. The goal is to obtain a solution to within a given error tolerance in the fastest possible time. If needed, the team must be able to generate the mesh (which may be $10^9 - 10^{12}$ or more grid points) from a CAD geometry and solve the problem.

There are many examples in science where innovation has been spurred through competition. DARPA, for example, has used this model to great effect. These challenge problems will motivate mathematicians and computational scientists to work on relevant problems of importance to the DOE. A head to head comparison of different approaches will spur improvements and allow application scientists to quantitatively compare competing techniques. The teams will need to keep in mind the overall goal (e.g. minimizing the time to solution).

Challenge problems could be developed at multiple levels of complexity: this could be different levels of complexity in the physics and geometry but also could include algorithm robustness to faults etc. The challenge problems should try to address all aspects of the solution process from problem setup and grid generation to the outputting of results. The metrics for success should be based on the application requirements such as accuracy, time-to-solution, setup time, memory use, etc.

Design of challenge problems will not be easy. There must be a balance between sufficient complexity in the physics but not so complex that development of a code is too burdensome. Perhaps some initial effort should be placed in developing prototype software frameworks for the different physics applications that would then allow teams to substitute new algorithms into selected parts of the framework.

2 Multi-algorithm optimizations

Proposition 2. *We should investigate (multiple-program multiple-data, MPMD) algorithms that can optimally solve the collection of algorithms/tasks from multi-physics simulations rather than optimizing individual algorithms/tasks in isolation.*

Although much progress has been made on developing algorithms for individual components of a large multi-physics code (e.g. grid generation, linear solvers, non-linear solves, etc.), there are potentially large benefits to be gained from optimizing the simulation code as a whole.

Multi-physics problems such as fluid-structure interaction (FSI) problems can involve the solution of complex physics in multiple connected and overlapping domains. An FSI problem, for example, may involve multiple fluid domains and multiple solid domains. Current partitioned algorithms typically time-step the solution by advancing the solution physics on one domain at a time (single-program multiple-data, SPMD). Advancing the solution on domain d, $d = 1, 2, \ldots, N_d$, typically consists of multiple high-level *tasks* T_i^d, $i = 1, 2, \ldots M_d$, (e.g. updating the momentum equations, solving a Poisson equation, generating a new grid, etc.) with certain dependencies between tasks. Each high-level task may itself consist of many lower level tasks (e.g. a multigrid algorithm to solve the Poisson equation has multiple sub-tasks). Instead of developing optimal algorithms for each task in isolation, why not optimize the entire collection of tasks? This exposes increased task parallelism and presents more opportunity to overlap communication with computation and to use otherwise idle processors from one task to accomplish useful work on another task.

Here is a simple example. Suppose we have two independent multigrid pressure solves to perform (e.g., from two domains). For a fine grid with N^3 points, each coarser level, l, has 8 times fewer grids points, $N^3/8^l$, and there is often not enough work on coarser levels to keep the processors busy. If, instead, we solve the coarse grid corrections for multigrid problem I on, say 1/8 the number of processors, and use the other 7/8 processors to perform the fine grid computations on multigrid problem II, then we can reduce the idle time. Moreover, in this scenario we might be be able to use a more robust, but usually poorer scaling W-cycle, instead of the better scaling V-cycle, thus obtaining an overall more robust and better scaling composite algorithm. Similarly, during communication phases in multigrid problem I, we could perform useful work on multigrid problem II.

In order to optimize across multiple algorithms we will need some of the following:

1. a scheduler than can allocate processors to tasks and schedule tasks.
2. algorithms (e.g. a flow solver, Krylov solver, or multigrid solver) will need to expose their sub-tasks (e.g. computational and communication tasks) so that they can be executed by the scheduler.
3. Tasks should provide performance models to the scheduler to indicate how long the task will take as a function of the number of processors and other machine properties.